Genuinely Polynominal Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem

by

James B. Orlin

Sloan W.P. No. 1615-84 December 1984

Genuinely Polynominal Simplex and Non-Simplex Algorithms for the Minimum Cost Flow Problem

by

James B. Orlin

Sloan W.P. No. 1615-84 December 1984

Abstract

We consider the minimum cost network flow problem min(cx : Ax=b, $x \geq 0$) on a graph G = (V,E). First we give a minor modification of Edmonds-Karp scaling technique, and we show that it solves the minimum cost flow problem in $O((|V|^2 \log |V|)(|E| + |V| \log |V|))$ steps. We also provide two dual simplex algorithms that solve the minimum cost flow problem in $O(|V|^4 \log |V|)$ pivots and $O(|V|^3 \log |V|)$ pivots respectively. Moreover, this latter dual simplex algorithm may be implemented so that the running time is proportional to that of Edmonds-Karp scaling technique.

Key words: network flow, scaling, simplex algorithm, polynomial algorithm.

Introduction

We consider the minimum cost network flow problem (1)

$$\begin{aligned} &\text{Minimize} \quad cx \\ &\text{Subject to } Ax = b \\ &\qquad\qquad x \geq 0, \end{aligned} \tag{1}$$

where A is the vertex-edge incidence matrix for a graph $G = (V,E)$. In addition, we assume that $V = \{0,1,2,\ldots,m\}$, c is an integral n-vector, and b is a real-valued m-vector with $-1 < b_i \leq 1$ for $i \in V$. We have deleted the redundant mass balance constraint for vertex 0. We assume without loss of generality that $|b_i| \leq 1$ since we may scale the variables without altering the optimum solution.

Let BIT(b) be the minimum integral value of s such that $2^s b$ is integer valued. Thus BIT(b) is a sufficient number of bits to represent b_i for all $i \in V$. We let BIT(b) = ∞ if no such finite value of s exists. We let BIT(c) = $\max\left(\lceil \log(|c_j|+1) \rceil : 1 \leq j \leq n\right)$.

We present three algorithms for the minimum cost network flow problem. The first algorithm is a modification of the original scaling algorithm of Edmonds and Karp (1972). This algorithm solves the minimum cost flow problem as a sequence of $O(|V| \text{BIT}(b))$ different shortest path problems, each of which may be solved in $O(|E| + |V| \log |V|)$ arithmetic steps using Fredman and Tarjan's (1984) implementation of Dijkstra's algorithms. In the case that BIT(b) is large (possibly infinite), we show how to reduce the number of shortest path problems to $O(|V|^2 \log |V|)$. Thus we show that the Edmonds-Karp procedure is in fact genuinely polynomial, i.e., the number of arithmetic operations is independent of BIT(b) or BIT(c). (The arithmetic steps used by the scaling algorithm are as follows: addition, subtractions, comparisons, truncation, and computing the smallest s such that $|2^s b_i| \geq 1$, i.e., calculating the place of the first non-zero bit in the binary representation of b_i.)

We also present two genuinely polynomial dual simplex algorithms for the minimum cost network flow problem. The first of these dual simplex algorithms takes $O(|V|^3 \mathrm{BIT}(b))$ pivots, or $O(|V|^4 \log |V|)$ pivots, whichever is smaller. The second dual simplex algorithm takes $O(|V|^2 \mathrm{BIT}(b))$ pivots or $O(|V|^3 \log |V|)$ pivots, whichever is smaller. The first of the dual simplex algorithms is a "more natural" pivot rule, and the proofs of the computational bounds are simpler. However, this latter dual simplex algorithm has the property that each dual pivot may be obtained via a (non-dual) Dijkstra step. Thus one may implement the latter dual simplex algorithm so that the number of arithmetic steps is $O(U^*(|E| + |V| \log |V|))$, where $U^* = \min(|V|\mathrm{BIT}(b), |V|^2)$. In this case, the number of arithmetic steps for the dual simplex algorithm is comparable to the number of arithmetic steps for the Edmonds-Karp scaling procedure. This is also the best known computational bound for the minimum cost network flow problem for sparce networks. Moreover, the dual simplex algorithms presented here are the first simplex pivot rules that are provably polynomial for the minimum cost network flow problem.

1. Background

Edmonds and Karp (1972) were the first to solve the minimum cost network flow problem in polynomial time. Their algorithm, now commonly referred to the Edmonds-Karp scaling technique, is to solve a sequence of $O(\mathrm{BIT}(b))$ different network flow problems, each equivalent to (1) except that the j-th such network flow problem has a right hand side of $\lceil 2^j b \rceil / 2^j$ rather than b. They also show that the j th problem can be solved from the (j-1) th problem via a sequence of at most $|V|$ shortest path problems. We will describe our implementation of Edmonds-Karp scaling technique in Section 2.

Although Edmonds and Karp did resolve the question of whether network flow problems can be solved in polynomial time, two interesting closely related questions were unresolved. First, as stated in their paper,

"A challenging open problem is to give a method for the minimum cost flow problem having a bound of computation which is polynomial in the number of nodes, and is independent of both costs and capacities".

We shall refer to such an algorithm as a genuinely polynomial algorithm. The reader should be forewarned that we are using the term "genuinely polynomial" in a slightly different sense than did Megiddo (1981). In particular, our calculations may be on real numbers, and we are permitting a different set of arithmetic operations than did Megiddo.

This first question is motivated in part by the existance of genuinely polynomial algorithms for several important subclasses of network flow problems, viz., the assignment problem, the shortest path problem, and the maximum flow problem.

The second question is as follows. Is there a simplex pivot rule that solves the minimum cost network flow problem in polynomial time? This latter question is motivated in part by the practical efficiency of the network simplex algorithm, as documented for example by Glover and Klingman (1975), and Ali et al. (1978). The question is motivated also by the recent average case results for the network simplex algorithm as proved by several researchers. See Karp et al. (1984) for a list of references.

Tardos (1984a) resolved the first of these two open questions. She showed how to solve the minimum cost flow problems by solving $|E|$ distinct problems such that in each problem $BIT(c) \leq 2 \log |V|$. Thus Edmonds-Karp scaling technique is genuinely polynomial for each of these $|E|$ problems. Tardos (1984b) shows how to extend her own technique to provide genuinely polynomial algorithms for all linear programs in which the constraint matrix coefficients are small, i.e., BIT(A) is polynomially bounded in m and n.

As for the second question, Zadeh (1973) provided the first negative evidence by showing that the primal simplex algorithm using Dantzig's pivot rule (i.e., pivot in the variable whose reduced cost is minimum) does take an exponentially large number of pivots in the worst case. Subsequently, Cunningham (1979) showed that Bland's primal simplex pivot rule takes an exponentially long sequence of consecutive degenerate pivots in the worst case. Indeed, Cunningham's example (a modification of an example of Edmonds (1970)) shows that Bland's rule takes an exponentially large number of pivots in the worst case even when specialized to the shortest path problem.

Cunningham (1979) also provided some "good news" with respect to the second question by developing a primal network simplex pivot rule that avoids "stalling", i.e., the number of consecutive degenerate pivots is polynomially bounded. Subsequently, Roohy-Laleh (1980), Balinski (1982), and Hung (1983) developed polynomial time simplex pivot rules for the assignment problem. Orlin (1984) showed that the number of pivots for Dantzig's pivot rule is $O(|V|^2|E|^2 BIT(b)\ 2^{BIT(b)})$. Thus when $2^{BIT(b)}$ is small — as it is the assignment problem and for the shortest path problem — Dantzig's pivot rule is polynomial time.

Ikura and Nemhauser (1983) developed a dual simplex pivot rule such that the number of pivots is polynomially bounded in $|V|$ and $2^{BIT(b)}$. Used in conjunction with Edmonds-Karp scaling, their algorithm solves the minimum cost network network flow problem in polynomial time.

We present the first specialization of the dual simplex algorithm which runs in polynomial time for network flow problems. It is still an interesting open question as to whether there is a "natural" primal simplex pivot rule which runs in polynomial time for network flow problems.

We observe that Balinski (1982), (1983) and (1984) in his work on the assignment problem and the transportation problem has provided some intriguing evidence which suggests why the dual polyhedra of network flow problems may be better suited for the simplex algorithm than the primal polyhedra. In particular, he shows that the number of vertices of the dual polyhedra is considerably smaller than for the primal polyhedra. He also shows that the Hersh conjecture is true when specialized to the dual network polyhedra. In addition, he provides a dual simplex procedure for the assignment problem for which the number of pivots is at most $(n^2-n)/2$, and he proves that this bound is the best possible.

The outline of the remainder of this paper is as follows. In Section 2, we review Edmonds-Karp scaling procedure, and we show that with a minor modification it is a genuinely polynomial algorithm. In Section 3, we present the first of our network dual simplex algorithms, and we show that the number of pivots is $O(|V|^4 \log |V|)$. In Section 4, we present the second of our network dual simplex algorithms and we show that the number of pivots is $O(|V|^3 \log |V|)$. Finally, in Section 5, we show how to implement this second dual simplex algorithm so that the number of arithmetic steps is $O(U^*(|E| + |V| \log |V|))$ steps, where $U^* = \min(|V|\mathrm{BIT}(b), |V|^2 \log |V|)$.

2. Edmonds-Karp Scaling Technique

In this section, we describe an implementation of Edmonds-Karp scaling technique for which the number of arithmetic operations is polynomially bounded in $|V|$.

Before describing our procedure, we first describe some of the notation and terminology that we will use. We also will make some simplifying assumptions.

Notation and Definitions

Let $G = (V,E)$ with $V = \{0,1,\ldots,m\}$. A path in G is an alternating sequence $P = v_0,e_1,v_1,\ldots,e_k,v_k$ of vertices and edges such that $e_i = (v_{i-1},v_i)$ or else $e_i = (v_i,v_{i-1})$. In the former case, e_i is called a forward edge of P. In the latter case it is called a backward edge of P. The cost of a path P is the sum of the costs of the forward edge of P minus the sum of the costs of the backward edges of P. A circuit is a path in which $v_0 = v_n$ and $v_0,\ldots,v_{n-1}$ are all distinct.

A directed path is a path in which every edge is a forward edge. A graph is strongly connected if there is a directed path between every pair of vertices.

By a rooted tree we mean a spanning tree in which one vertex is specified to be the root. Unless specified otherwise, we will henceforth assume that the root vertex of any rooted tree is vertex 0.

Let T be a rooted tree. For every pair u,v of vertices, we let $P^T(u,v)$ denote the unique path in T from vertex u to vertex v. For every vertex u and every edge $e \in T$ we let $P^T(u,e)$ denote the path in T whose initial vertex is u and whose terminal edge is e. We say that $e \in T$ is a downward edge ofthe rooted tree T if e is a forward edge of $P^T(0,e)$. Otherwise, e is an upward edge of T. We let

$$B(T,e) = \{i \in V : e \in P^T(0,i)\}.$$

Equivalently, B(T,e) is the set of vertices of V that lie below e on the tree T.

For a tree T and each $e \notin T$, we let c_e^T denote the cost of the circuit C created by adding edge e to T, where e is a forward edge of C. Equivalently, c_e^T is the reduced cost of edge e with respect to the basis induced by T.

For any pair S^1, S^2 of disjoint sets of vertices, we let

$$\delta(S^1,S^2) = \{(u,v) \in E : u \in S^1,\ v \in S^2\}.$$

For any edge e of T we define the fundamental cutset of e to be

$$F(T,e) = \begin{cases} (u,v) \in E : u \not\in B(T,e),\ v \in B(T,e) & \text{if } e \text{ is upward} \\ (u,v) \in E : u \in B(T,e),\ v \in B(T,e) & \text{if } e \text{ is downward.} \end{cases}$$

<u>The Parametric Sequence of Problems</u>

For each $\ell = 0,1,2,\ldots$, we define $b(\ell)$ as follows.
Let $s = \lfloor(\ell-1)/m\rfloor$ and let $v = \ell - sm$. Then

$$b_i(\ell) = \begin{cases} \lceil 2^s b_i \rceil / 2^s & \text{if } 1 \le i \le r \\ \lceil 2^{s-1} b_i \rceil / ^{s-1} & \text{if } r+1 \le i \le m. \end{cases}$$

We let PROB(ℓ) denote the following problem.

$$\begin{array}{ll} \text{Minimize} & cx \\ \text{Subject to} & Ax = b(\ell) \\ & x \geq 0. \end{array} \qquad \text{PROB}(\ell)$$

We first observe that $b_i(0) = 1$ or 0 according as b_i is positive or not. Thus PROB(0) is equivalent to finding the shortest path from vertex i to vertex 0 for all i with $b_i > 0$.

We also observe that for $U^* = (m+1)\,\text{BIT}(b)$, PROB($U^*$) is the same as problem (1). Henceforth we will refer to (1) as PROB(∞).

REMARK 1. Suppose that $b(\ell+1) \neq b(\ell)$, and that $\ell+1 = sm+r$ with $s = \lfloor \ell/m \rfloor$. Then

$$b_i(\ell+1) - b_i(\ell) = \begin{cases} -2^{-s} & \text{if } i = r \\ 0 & \text{if } i \neq r. \end{cases} \qquad \square$$

Remark 1 follows directly from the definition of $b(\ell)$. We observe also that $b(\ell)$ is monotonically decreasing in ℓ. Although the monotonicity of b(.) is not critical to the usual implementation of Edmonds-Karp scaling technique, the monotonicity of b(.) is required in the proofs of Lemmas 3, 4, and 5 below.

Edmonds-Karp scaling algorithm may be summarized as follows:

STEP 1. Solve PROB(0) by solving a shortest path problem.

STEP 2. For $\ell = 0$ to $(m+1)\text{BIT}(b)$, solve PROB($\ell+1$) as a shortest path problem as derived from the optimal solution to PROB(ℓ).

We will explain Step 2 in more detail below; however, in order to simplify the subsequent description and analysis we first make the following simplifying assumptions.

A1. The graph $G = (V,E)$ is strongly connected.

A2. The matrix A has full row rank.

A3. For any directed circuit C of G, the cost of C is nonnegative.

A4. For any circuit C of G, the cost of C is non-zero.

A5. $\mathrm{BIT}(b) < \infty$.

We first show that we may make assumptions A1-A4 without loss of generality.

If G is not strongly connected, then we may add artificial adges $(0,j)$ and $(j,0)$ for each $j \in [1..m]$, each edge with a suitable large cost. One of these edges would have positive flow in an optimum solution for PROB(ℓ) if and only if PROB(ℓ) had no feasible solution without flows in artificial edges. As for A2, we have previously assumed that we eliminated the redundant supply/demand constraint for vertex 0. In conjunction with asssumption A1, it follows that A has full row rank.

A3 is equivalent to dual feasibility. If there is a negative cost directed circuit, then there is either no feasible solution to PROB(ℓ) or else PROB(ℓ) is unbounded.

To achieve A4, we may add ε^{-j} to the cost of the jth edge of E. Equivalently, we solve the network flow problems described below using lexicography.

Assumption (A5) is not without loss of generality, but we will relax this assumption later in this section.

LEMMA 1. Suppose that the data for PROB(ℓ) satisfy A1, A2, and A4. Then there is a unique optimum solution for PROB(ℓ) for each $\ell = 0,1,2,\ldots$

PROOF. Assumption A1 guarantees primal feasibility of PROB(ℓ). Assumption A2 guarantees dual feasibility. Assumption A4 guarantees dual non-degeneracy. Therefore, there is always a unique optimal solution. □

Solving PROB(ℓ+1)

For any spanning tree T and for any integer $\ell \geq 0$, we let $x^T(\ell)$ denote the basic (possible infeasible) solution obtained for PROB(ℓ) with basis T. We say that T is optimal for PROB(ℓ) if $x^T(\ell) \geq 0$ and $c^T \geq 0$.

Suppose that the spanning tree T is optimal for PROB(ℓ) but not for PROB(ℓ+1). We construct the auxiliary graph $\hat{G} = (\hat{V},\hat{E})$ with costs $\hat{c}$ for PROB(ℓ+1) from T as follows. The vertex set is $\hat{V} = V$. For each edge $e = (i,j)$ with $x_e^T(\ell) = 0$, there is a corresponding edge $(i,j) \in \hat{E}$ with $\hat{c}_{ij} = c_{ij}^T$. For each edge $e = (i,j)$ with $x_e^T(\ell) > 0$, there are two corresponding edges (i,j) and (j,i) in $\hat{E}$ with $\hat{c}_{ij} = \hat{c}_{ji} = 0$.

By the shortest path problem for $\hat{G}$, we mean the problem of finding the shortest path from vertex 0 to every other vertex of $\hat{G}$.

LEMMA 2. Suppose that the spanning tree T is optimal for PROB(ℓ) but not for PROB(ℓ+1), with $\ell+1 = sm+r$. Let $\hat{G}$ be the auxiliary graph for PROB(ℓ) constructed from T. Let S be an optimal spanning tree for the shortest path problem for $\hat{G}$, and let $\hat{x} = x^S(\ell+1)$. Then

(i) S is an optimal basis for PROB(ℓ+1).

(ii) $\hat{x}$ is the unique optimal solution for PROB(ℓ+1).

(iii) $|\hat{x}_e - x_e(\ell)| = 0$ or 2^{-s} for each $e \in E$.

PROOF. By assumption A1 and by the dual feasibility of T, there is some optimal shortest path tree. Edmonds and Karp showed that (i) is true. The uniqueness of $\hat{x}$ follows from assumption A4.

Finally, by Remark 1, $2^s(\hat{x}_e - x_e^T)$ is integer valued. In fact, $\hat{x}$ obtained from x^T by sending 2^{-s} units of flow along the cheapest path in the auxiliary graph. □

Let $\quad \text{REM}(\ell) = \sum_{i=1}^{m}\big(b_i(\ell) - b_i\big).$

The following corollary of Lemma 1 will be important in the proof that Edmonds-Karp scaling algorithm is genuinely polynomial.

COROLLARY 1. Suppose that $x(\ell_1)$ and $x(\ell_2)$ are optimal solutions for PROB(ℓ_1) and PROB(ℓ_2) respectively. Then for each edge $e \in E$,

$$|x_e(\ell_1) - x_e(\ell_2)| \le |\text{REM}(\ell_1) - \text{REM}(\ell_2)|.$$

PROOF. Suppose that $\ell_1 \le \ell_2$. Then

$$\begin{aligned} |x_e(\ell_1) - x_e(\ell_2)| &\le \sum_{k=\ell_1}^{\ell_2-1} |x_e(k+1) - x_e(k)| \\ &\le \sum_{k=\ell_1}^{\ell_2-1} \big(\text{REM}(k) - \text{REM}(k+1)\big) \\ &= \text{REM}(\ell_1) - \text{REM}(\ell_2), \end{aligned}$$

with the second inequality being a consequence of (iii) of lemma 2. □

We now present a genuinely polynomial version of Edmonds-Karp scaling algorithm.

<u>Edmonds-Karp Scaling Algorithm</u>

```
Begin
    Solve the problem of finding a shortest path from every vertex
    $i \in [1..m]$ to vertex 0 in graph G. Let T be the optimum spanning tree.

While T is not optimum for PROB($\infty$) do
    begin
        find the largest value of $\ell$ such that T is optimal for PROB($\ell$);
        construct the auxiliary graph $\hat{G}$ for PROB($\ell+1$) from T;
        let $\hat{T}$ be the solution to the shortest path problem in $\hat{G}$;
        let T = $\hat{T}$;
    end
end.
```

The proof of the genuine polynomiality of the Edmonds-Karp algorithm will rely on two aspects. First, we will show that we can find in polynomial time the largest value ℓ such that T is optimal for PROB(ℓ). Second, we will show that the number of distinct trees obtained by the algorithm in the "while loop" is $O(m^2 \log m)$.

Henceforth, we will let SUPFEAS(T) denote the largest value of ℓ such that T is primal feasible for PROB(ℓ). If $x^T(\infty) \geq 0$ then SUPFEAS(T) = ∞. If T is not feasible for PROB(ℓ) for any ℓ then SUPFEAS(T) = $-\infty$.

Our procedure for calculating SUPFEAS(T) relies on Lemmas 3 and 5 below.

Lemma 3 show that we can calculate SUPFEAS(T) using binary search. Lemma 5 shows that we may restrict the search interval for the value SUPFEAS(T) to an interval of size O(m log m).

LEMMA 3. Let T be a spanning tree and let e be an edge of T. Let $\ell_1, \ell_2 \in Z$ with $\ell_1 < \ell_2$.

(i) If e is an upward edge of T, then $x_e^T(\ell_1) \geq x_e^T(\ell_2)$.

(ii) If e is a downward edge of T, then $x_e^T(\ell_1) \leq x_e^T(\ell_2)$.

PROOF. If e is an upward edge of T, then

$$x_e^T(\ell) = \sum_{i \in B(T,e)} b_i(\ell).$$

Thus (i) follows from the fact that $b_i(\ell)$ is monotonically non-increasing in ℓ.

If e is a downward edge of T, then

$$x_e^T(\ell) = -\sum_{i \in B(T,e)} b_i(\ell),$$

and (ii) follows from the monotonicity of $b_i(\ell)$. □

LEMMA 4. Let T be a rooted tree and suppose that ℓ = SUPFEAS(T) with $\ell < \infty$. Let a be an edge such that $x_a^T(\ell+1) < 0$, and let $s = \lfloor \ell/m \rfloor$. Then

(i) $x_a^T(\ell) = 0$.

(ii) Edge a is an upward edge of T.

(iii) $x_a^T(\ell+1) = -2^{-s}$.

PROOF. We first observe that $2^s b(\ell+1)$ is integer valued, and thus $2^s x^T(\ell+1)$ and $2^s x^T(\ell)$ are both integral. Therefore,

$$2^s x_a^T(\ell+1) + 1 \leq 0 \leq 2^s x_a^T(\ell). \quad (2)$$

Then (i) and (ii) follow from (2) together with (iii) of Lemma 2.

Because $x_a^T(\ell) > x_a^T(\ell+1)$, it follows from Lemma 3 that edge a is upward. □

LEMMA 5. Let T be a rooted tree and let e be an upward edge of T for which $x_e^T(\infty) < 0$. Let $s = -\lfloor \log(-x_e^T(\infty)) \rfloor$. Then

$$x_e^T(sm) \geq 0 > x_e^T(m(s+2 + \lceil \log m \rceil).$$

PROOF. Suppose first that $x_e^T(sm) < 0$. Since $2^s x^T(sm)$ is integer, it follows that $x_e^T(sm) \leq -2^{-s}$. Since e is an upward edge, we know that $x_e^T(\infty) \leq x_e^T(sm) \leq -2^{-s}$, contradicting our definition of s.

Consider next $\ell' = m(s+2+\lceil \log m \rceil)$. By Corollary 1, $x_e^T(\ell') - x_e^T(\infty) \leq \text{REM}(\ell')$. Moreover,

$$\text{REM}(\ell') < 2^{-s+1}$$

and thus $x_e^T(\ell') < 0$ by our choice of s. □

By Lemma 3, we know that the set of integers ℓ for which $x^T(\ell) \geq 0$ is an interval. Therefore, to find SUPFEAS(T), we only have to find an integer ℓ for which $x^T(\ell) \geq 0$ and $x_a^T(\ell+1) < 0$ for some upward edge a if T. By Lemma 5, we may restrict our search for the value SUPFEAS(T) to a range depending on $x_a^T(\infty)$. We combine these properties so as to obtain the following procedure for computing SUPFEAS(T).

PROCEDURE 1. "Compute SUPFEAS(T)"

```
Begin
    Let $q = \min(x_a^T(\infty)$: a is an upward edge of T;
    if $q \geq 0$, then let $\ell = \infty$;
    else begin
        let $s = -\lfloor \log - q \rfloor$;
        let $\ell_1 = sm$;
        let $\ell_2 = \ell_1 + m(2+\log m)$;
        use binary search to find a value $\ell$ in $[\ell_1..\ell_2]$ such that
         $x_a^T(\ell) \geq 0$ for all upward edges a of T and $x_a^T(\ell+1) < 0$ for some
        downward edge a of T;
    end
    if $x^T(\ell) \geq 0$ then let SUPFEAS(T) = $\ell$;
    else let SUPFEAS(T) = $-\infty$;
end.
```

PROPOSITION 1. Procedure 1 computes the correct value of SUPFEAS(T) in $O(|V| \log |V|)$ steps.

PROOF. Let q, s, ℓ_1, ℓ_2 and ℓ be defined as in Procedure 1, and let e be an upward edge of T for which $x_e^T(\infty) = q$.
Let ℓ^* be the maximum value such that $x_a^T(\ell^*) \geq 0$ for each upward edge a of T.
We first note that if $q \geq 0$ then $\ell^* = \ell = \infty$ as in the procedure. Otherwise, by

Lemma 5, $\ell^* \in [\ell_1 .. \ell_2]$. Moreover, by Lemma 3, the value ℓ computed by binary search is equal to ℓ^*.

If $x_a^T(\ell) < 0$ for some a, then a must be a downward edge and thus by Lemma 3, $x_a^T(\ell') < 0$ for all $\ell' < \ell$. In this case SUPFEAS(T) = $-\infty$. Otherwise $x^T(\ell) \geq 0$, and by our choice of ℓ, $x^T(\ell') \geq 0$ for $\ell' > \ell$. Thus ℓ = SUPERFEAS(T).

We also note that binary search over the interval $[\ell_1 .. \ell_2]$ takes $O(\log(\ell_2 - \ell_1))$ "tests" where a "test" consists in checking the feasibility of $x^T(\ell')$ for some ℓ'. Since each "test" takes $O(|V|)$ steps, the procedure takes $O(|V| \log |V|)$ steps. □

In the remainder of this section, we wish to show that the number of iterations of the "while loop" of Algorithm 1 is polynomially bounded. Equivalently, we wish to show that the number of distinct trees determined by the algorithm is polynomially bounded.

We first let $\text{PERM}(\ell) = \{e \in E : x_e(\ell) > \text{REM}(\ell)\}$.

By Corollary 1, we know that each edge $e \in \text{PERM}(\ell)$ is such that $x_e(\ell') > 0$ for all $\ell' > \ell$. We also know from Corollary 1 that $\text{PERM}(\ell) \subseteq \text{PERM}(\ell+1)$.

LEMMA 6. Let T be a rooted tree that is optimal for PROB(ℓ). Let ℓ' = SUPFEAS(T). Then

$$\text{PERM}(\ell') \subsetneq \text{PERM}(\ell' + m + 2m\lceil \log m \rceil).$$

PROOF. Let $\ell^* = \ell' + m + 2m\lceil \log m \rceil$. Let e be an edge of T such that $x_e^T(\ell'+1) < 0$. Let $B = B(T,e)$, and let $\bar{B} = V - B$.

By Lemma 3, we know that e is an upward edge of T and thus $T \cap \delta(\bar{B},B) = \phi$. We will show that there is an edge $\alpha \in \delta(\bar{B},B)$ such that $\alpha \in PERM(\ell^*)$. Since $PERM(\ell') \subseteq PERM(\ell^*)$ this will complete the proof. □

Let $s = \lfloor(\ell'-1)/m\rfloor$. Then

$$\sum_{a\in\delta(B,\bar{B})} x_a(\ell^*) - \sum_{a\in\delta(\bar{B},B)} x_a(\ell^*) = -\sum_{i\in B} b_i(\ell^*) \geq -\sum_{i\in B} b_i(\ell') \geq 2^{-s}. \quad (3)$$

By (3) and the non-negativity of $x(\ell^*)$, there is some edge $\alpha \in \delta(\bar{B},B)$ such that

$$x_\alpha(\ell^*) \geq 2^{-s}/m.$$

Finally, we observe that $REM(\ell^*) < m2^{-s+2\log m} \leq 2^{-s}/m$, and thus $x_\alpha(\ell^*) > REM(\ell^*)$, completing the proof.

THEOREM 1. The Edmonds-Karp scaling algorithm solves the minimum cost network flow problem with $O(U^*(|V|\log|V| + |E|))$ arithmetic operations, where $U^* = \min(|V|^2\log |V|, |V| \, BIT(b))$.

PROOF. Let $T^1,\ldots,T^t$ be the distinct trees determined by Algorithm 1, and let $\ell^i = SUPFEAS\ (T^i)$. To compute T^i takes $O(|V|\ |E|)$ steps using a shortest path procedure. If we are given T^i, we can compute ℓ^i in $O(|V| \log |V)$ steps by Procedure 1. We can also compute T^{i+1} in $O(|E| + |V| \log |V|)$ steps using the Fredman-Tarjan (1984) data structure of fibonacci heaps to implement Dijkstra's algorithm. (We use the reduced costs c^T where $T = T^i$, so that the costs on the auxiliary graph are nonnegative.) Moreover, it is clear that $t \leq |V|\ BIT(b)$. To complete the proof of Theorem 1, it suffices to show that

$t \leq 2|V|^2 \log |V| + |V|^2$.

Let $S_k = \{\ell^i : i \in [i..t] \text{ and } \mathrm{PERM}(\ell^i) = k\}$. By Lemma 6, $|S_k| \leq 2|V|\ (1+ \log |V|)$ and hence $t \leq 2|V|^2 \log |V| + |V|^2$ completing the proof.

3. A Genuinely Polynomial Dual Simplex Pivot Rule for the Minimum Cost Network Flow Problem.

In this section we develop the first of two pivot rules for solving the minimum cost network flow problems. Both rules appear to be "parametric rules" in the following sense: we will show how to pivot so as to obtain an optimal basis for PROB(ℓ+1) starting from an optimal basis for PROB(ℓ). (Thus we will essentially solve the shortest path problem on the auxiliary graph by a sequence of dual pivots.) Despite the fact that the pivot rule appears to be defined parametrically, we will show that both of these pivot rules are, in fact, dual simplex pivot rules for the original problems, PROB(∞).

A Dual Simplex Pivot Rule

In linear programming, the dual simplex pivot rule may be summarized as follows. Given a dual feasible basis B and an infeasible basic solution x (where $x_B = B^{-1}b$), pivot out a basic variable x_i with $x_i < 0$ and pivot in a variable so that the resulting pivot results in a dual feasible basis. Within the context of network flows, if x^T is the current basic solution and if x_a is the exiting variable for $a \in E$, then the entering variable in x_e where $e \in F(T,e)$ is chosen so $c_e^T = \min(c_\alpha^T : \alpha \in F(T,e))$. (Recall that F(T,e) is the fundamental cutset induced by e.)

The algorithm presented below is a refinement of the dual simplex algorithm in that we choose a unique edge with $x_a^T < 0$ to leave the basis.

ALGORITHM 2. "The Scaling Dual Simplex Algorithm".

```
Begin
    Use a phase 1 procedure to find an optimal spanning tree T with respect
    to PROB(0). (We will discuss the phase 1 approach in the following
    subsection.)

    While T is not optimal for PROB($\infty$) do
    begin
        let $\ell$ = SUPFEAS(T);
        Let $a \in T$ be chosen so that $x_a^T(\ell+1) < 0$,
        and $x_\alpha^T(\ell+1) \geq 0$ for each other edge $\alpha$ on the path $P^T(0,a)$;
        let T' be obtained from T by pivoting out edge a and pivoting in the
        edge $e \in F(T,e)$ with $c_e^T = \min(c_\alpha^T : \alpha \in F(T,a))$;
        let T = T';
    end
end.
```

We will discuss the phase 1 procedure subsequent to discussing the number of pivots subsequent to solving PROB(0).

THEOREM 2. Algorithm 2 is a genuinely polynomial dual simplex algorithm for the minimum cost network flow problem. The number of dual simplex pivots starting from an optimal tree for PROB(0) is $O(|V|^2 U^*)$, where $U^* = \min(|V|^2 \log |V|), |V| \text{ BIT}(b))$.

PROOF. Assume first that T is a dual feasible basis determined by the algorithm and that ℓ = SUPFEAS(T) with $\ell > -\infty$. This is certainly true for an optimal solution to PROB(0). Let S be the next basis determined by the algorithm. We will show that the pivot is a dual simplex pivot and that SUPFEAS(T) $\leq$ SUPFEAS(S).

Let ℓ = SUPFEAS(T). By (ii) of Lemma 4, the edge a pivoted out of T is upward, and $x_a^T(\ell) = 0$. Moreover, $x_a^T(\infty) \leq x_a^T(\ell+1) < 0$, and thus the pivot is a dual simplex pivot.

Since $x_a^T(\ell) = 0$, the pivot from T to S is a degenerate pivot with respect to PROB(ℓ). The degeneracy implies that $x^S(\ell) = x^T(\ell) \geq 0$ and thus SUPFEAS(S) $\geq \ell$.

Let $T^1, T^2, \ldots, T^t$ be the set of trees determined by the algorithm and let ℓ^i = SUPFEAS(T^i). We have already shown in the proof of Theorem 1 that the maximum number of distinct values of ℓ^i is O(U*). To complete the proof of the Theorem we will show that if SUPFEAS(T^i) = SUPFEAS(T^j) then $j \leq i+|V|^2$.

Let $T^i, \ldots, T^j$ be a set of trees for which SUPFEAS(T^i) = SUPFEAS(T^j) = ℓ. Without loss of generality, we assume that $x_e(\ell) = 0$ for all $e \in E$. Otherwise, we would contract the edges a of E for which $x_a(\ell) > 0$ (and thus $c_a^T(\ell) = 0$ for $T = T^i, \ldots, T^j$.) This contraction does not effect either the choice of the entering variable or the choice of the exiting variable in any of the pivots of $T^i, \ldots, T^{j-1}$.

Let $T = T^k$ for some $k \in [i..j]$. We say that a vertex v of T is <u>green</u> if the path $P^T(0,v)$ from 0 to v contains no upward edges. We let G^k denote the set of green vertices of T^k.

We let $d^k(v)$ denote the indegree of vertex v in the tree T^k. We let S^k denote the set of non-root vertices of T^1 with indegree = 0. Thus S^k is the set of sources of T^k. The fact that $j-i \leq |V|^2$ is a consequence of the following Lemma.

LEMMA 7. For each $k \in [i..j-1]$

(i) $G^k \subseteq G^{k+1}$.

(ii) $\sum_{v \in G^k} d^k(v) = \sum_{v \in G^k} d^{k+1}(v) + 1$.

(iii) $S^{k+1} \subseteq S^k$.

(iv) $|G^k| < \sum_{v \in G^k} d^i(v) \leq |G^k| + |S^k| - 1$.

We first show why Lemma 7 implies that $j-i \leq |V|^2$. We first note that by(i) there are at most $|V|$ pivots for which $|G^{k+1}|$ is greater than $|G^k|$. By (ii) if $G^k = G^p$ with $k < p$, then

$$\sum_{v \in G^k} \left(d^k(v) - d^p(v)\right) = p-k,$$

and by (iv) it follows that $p \leq k + |S^k| \leq k+|V|$. Thus the numbers of consecutive pivots for which $|G^{k+1}| = |G^k|$ is at most $|V|$, and thus $j-i \leq |V|^2$.

PROOF OF LEMMA 7. Let us denote T^k and T^{k+1} as T and S respectively. Let a be the edge pivoted out of T and let e be the edge pivoted into S.

We first prove (i). Let v be any vertex of G^k. Since v is green we know that a is not an edge on the path $P = P^T(0,v)$. Therefore P is also a path in S, from which it follows that v is green in S. Therefore $G^k \subseteq G^{k+1}$.

We next prove (ii). Let v be the head of edge a and let u be the head of edge e. The exiting variable rule for our dual simplex algorithm guarantees that $v \in G^k$. The entering variable rule guarantees that $u \notin G^k$. Thus in pivoting from T to S we delete the edge a whose head is in G^k and we add an edge e whose head is not in G^k. Therefore (ii) is true.

We next prove (iii). Let v be the head of edge a as defined as in the proof of (ii). Since v is green and we have assumed that $x(\ell) = 0$, it follows that the path $P^T(0,v)$ consists of downward edges. From this fact we conclude that $v = 0$ or else $d^k(v) > 0$. In either case, $v \notin S^k$. We have thus shown that $S^{k+1} \subseteq S^k$ since the pivot from T^k to T^{k+1} will not create any non-root sources.

We now prove (iv). Since each non-root vertex of G^k has indegree at least one,

$$\sum_{v \in G^k} d^k(v) \geq |G^k| - 1.$$

The sum of the indegrees of the remaining vertices is at least $|V| - |G^k| - |S^k|$ since this number equals the number of vertices not in G^k with indegree greater than 0. Because

$$\sum_{v \in T} d^k(v) = |V| - 1,$$

it follows that

$$\sum_{v\in G^k} d^k(v) \leq (|V| - 1) - (|V| - |G^k| - |S^k|)$$
$$= |G^k| + |S^k| - 1,$$

completing the proof of Lemma 7 and Theorem 2. □

It is an interesting open question whether the worst case bound of $O(|V|^4 \log |V|)$ pivots is acheivable. I conjecture that the maximum number of pivots is $O(|V|^3 \log |V|)$, but I do not know of a proof of such a bound. It is also conceivable that the maximum number of pivots is significantly less.

5. A Phase 1 Procedure

We still have not yet specified our method for solving PROB(0). In this subsection we will present several alternatives.

If we permit ourselves algorithms other than the simplex algorithm, then we may solve PROB(0) using standard techniques. If $c \geq 0$, we may solve the shortest path problem in $O(|V| \log |V|+|E|)$ steps using Dijkstra's algorithm with the data structure Fibonacci heaps. If c is not non-negative, then we may solve the shortest problem in $O(|V||E|)$ steps using the label-correcting algorithm for shortest paths.

If we permit ourselves the use of the primal simplex algorithm, we also may still solve PROB(0) quite efficiently. If $c \geq 0$, then Dantzig's primal rule leads to the same sequence of pivots as does Dijkstra's algorithm, as proved independently by Zadeh (1979) and by Dial et al. (1979). If c is not non-negative, then we may still interpret a minor modification of the label correcting algorithm as a special case of the primal simplex algorithm. (See Cunningham (1979).)

In keeping with the spirit of the rest of this paper, we will show how to solve PROB(0) via a dual simplex algorithm. As in the case of the usual shortest path algorithms we will consider the cases of $c \geq 0$ and $c \ngeq 0$ separately.

Solving PROB(0) if the costs are non-negative.

Let $d(\ell)$ be defined as follows for $\ell \in [0..2m]$.

$$d_i(\ell) = \begin{cases} 1 & \text{if } \ell + 1 \leq i \\ 0 & \text{if } \ell - m + 1 \leq i \leq \ell \\ -1 & \text{if } \ell - 2m + 1 \leq i \leq \ell - m. \end{cases}$$

Let PROB1(ℓ) be defined as follows.

$$\begin{array}{lll} \text{Minimize} & cx & \\ \text{Subject to} & (-A)x = d(\ell) & \text{PROB1}(\ell) \\ & x \geq 0. & \end{array}$$

We observe that $-A$ is the vertex-edge icidence mature of the graph G' obtained G by reversing all edges. Thus PROB1(ℓ) is a minimum cost network flow problem satisfying assumptions A1-A5.

Suppose that we carry out our dual simplex algorithm to solve PROB1(2m) starting from the artificial tree T for PROB1(0), where T consists of the edges $\{(j,0) : j \in [1..m]\}$ each with a cost of 0. Theorem 2 shows that the number of pivots is $O(|V|^3)$. Moreover, the optimal basis T' for PROB1(2m) has no artificial arcs, and its reversal is optimal for PROB(0). To see this, note that T' is the tree in G' of shortest distances from the root vertex 0. Its reversal is the tree in G of shortest distance to the root vertex 0.

Solving PROB(0) if the costs may be negative.

Our dual implex procedure of the previous section relied on the fact that the original artificial basis was dual feasible for PROB(ℓ) for each ℓ. Unfortunately, in the case that c is not non-negative, there is no obvious artificial basis that is dual feasible.

In order to create a dual feasible basis, we use a standard method of linear programming of introducing an additional constraint, creating PROB2.

$$\begin{array}{ll} \text{Minimize} & cx \\ \text{Subject to} & Ax = 1 \\ & \sum_{e\in E} x_e + x_0 = M \\ & x \geq 0, \end{array} \qquad \text{PROB2}$$

where 1 denotes an m-vector with a 1 in each component, M is a suitably large integer, and x_0 is a slack variable (or in graph terms, it represents a loop at vertex 0).

This latter problem is solved by Karp and Orlin (1981) in $O(|V||E| \log |V|)$ steps via a dual simplex algorithm.

Actually, to be more precise, Karp and Orlin solve the parametric problem obtained by dualizing the "redundant" constraint as follows

$$\begin{array}{ll} \text{Minimize} & \sum(c_j-\lambda)x_j - \lambda x_0 \\ \text{Subject to} & Ax = 1 \\ & x \geq 0 \end{array} \qquad \text{PROB2}(\lambda)$$

as λ varies from $+\infty$ to 0. However, we leave it to the reader to show that under the assumption of dual non-degeneracy, moving from one breakpoint to the next breakpoint in the parametric simplex algorithm corresponds to executing a dual pivot simplex pivot with respect to PROB2.

4. A Second Dual Simplex Algorithm

We expect that the primary contribution of Algorithm 2 will be theoretical rather than computational. Computationally, Algorithm 2 has two significant drawbacks. First each dual simplex pivot may require $\Omega(|E|)$ steps. Second, the number of degenerate pivots with respect to PROB(ℓ) may be $\Omega(|V|^2)$.

In this section we offer a speed up technique so that the number of degenerate pivots with respect to PROB(ℓ) is at most $|V|$. Moreover, this dual simplex algorithm may be implemented so as to run in time proportional to the Edmonds-Karp scaling technique. In fact, it is equivalent in a very real sense to Edmonds-Karp scaling, as seen in Section 5.

The major idea of the algorithm is to try to enforce a condition so that the cardinality of S^k as described in the proof of Lemma 7 is at most 1.

To describe this dual simplex algorithm, we first define "strong feasibility", a concept introduced independently by Cunningham (1976) and Barr et. al (1977).

A tree T is said to be <u>strongly feasible</u> for PROB(ℓ) if $x^T(\ell) \geq 0$ and if $x_e^T(\ell) > 0$ for each upward edge e of T. We will say that a basis T is <u>strongly optimal</u> for PROB(ℓ) if T is both optimal and strongly feasible for PROB(ℓ).

In the following, we let SUPSTRONG(T) be the largest value ℓ such that T is strongly optimal for PROB(ℓ).

ALGORITHM 3. "The Modified Dual Simplex Scaling Algorithm".

```
Begin
    Use a phase 1 procedure to find a spanning tree T that is strongly
    optimal for PROB(0).

    While T is not strongly optimal for PROB($\infty$) do

    begin
        let $\ell$ = SUPSTRONG(T);
        let S = T;
        while S is not strongly feasible for PROB($\ell$+1) do
        begin
            let "a" be an upward edge of S for
            which $x_a^S(\ell+1) = 0$ and $x_e^S(\ell+1) > 0$
            for each upward edge e on the path from 0 to a in S;
            let $e \in F(S,a)$ be an edge with
             $c_e^S = \min\left(c_\alpha^S : \alpha \in F(T,a)\right)$;
            let S' be obtained by pivoting in edge e and pivoting out edge a.
            let S = S';
        end
        let T = S;
    end
end.
```

This algorithm is identical to Algorithm 2 except that we require our basis to be strongly optimal rather than optimal. As such, the proof of correctness of the algorithm and the proof of the polynomial bound are both similar to the corresponding proofs for Algorithm 2.

In the remainder of this section we treat three aspects of the algorithm. We first point out that the phase 1 procedure for Algorithm 3 is essentially the same as the corresponding procedure for Algorithm 2. Next we show how to calculate SUPSTRONG(T) in $O(|V|\log|V|)$ steps. Finally, we will show that the algorithm is correct and the number of pivots is $O(|V|^3\log|V|)$.

The Phase 1 Procedure

Here we may carry out the same phase 1 procedure as in Algorithm 2. The terminal basis is optimal for

$$\begin{aligned} &\text{Minimize} \quad cx \\ &\text{Subject to } Ax = 1 \\ &\qquad\qquad\quad x \geq 0. \end{aligned} \tag{4}$$

Moreover, each feasible basis is non-degenerate with respect to problem (4), and thus the terminal basis of phase 1 is strongly optimal for (4). We may now solve PROB(0) by solving at most $|V|$ intermediate problems starting with problem (4). At each step we would reduce the right hand side from 1 to 0 for some component i with $b_i(0) = 0$.

Computing SUPSTRONG

Despite the close connections between SUPSTRONG and SUPFEAS, it is not true that the computation of SUPSTRONG easily reduces to the computation of SUPFEAS. In particular, to compute SUPSTRONG in a polynomial number of arithmetics steps, it appears that we must be able to evaluate the expression BIT(α) in polynomial time.

If we count the evaluation of BIT(α) as one arithmetic step, then we may determine SUPSTRONG in $O(|V| \log |V|)$ steps. This algorithm is based on the following lemma.

LEMMA 8. Let T be a tree with root vertex 0, let ℓ = SUPSTRONG(T), and let ℓ' = SUPFEAS(T). Let e be an upward edge of T for which $x_e^T(\ell) = 0$. Then

$$\lfloor \ell/m \rfloor = \max\left(\mathrm{BIT}(b_i(\ell')) : i \in B(T,e)\right).$$

PROOF. We first note that $\ell \leq \ell'-1$ and that

$$\textstyle\sum_{i \in B(T,e)} b_i(\ell+1) = \sum_{i \in B(T,e)} b_i(\ell') = 0.$$

Thus $b_i(\ell+1) = b_i(\ell')$ for each $i \in B(T,a)$. In addition,

$$\textstyle\sum_{i \in B(T,a)} b_i(\ell) = 2^{-s},$$

where $s = \lfloor \ell/m \rfloor$. Therefore,

$$\begin{aligned} s = \mathrm{BIT}\big(b(\ell)\big) &= \max\left(\mathrm{BIT}(b_i(\ell+1) : i \in B(T,e)\right) \\ &= \max\left(\mathrm{BIT}(b_i(\ell') : i \in B(T,e)\right). \end{aligned}$$

We can thus solve for $\ell' = \text{SUPSTRONG}(T)$ in $O(|V| \log |V|)$ steps by first solving for SUPFEAS(T) and then using the results of Lemma 8 to compute $\lfloor \ell'/m \rfloor$. It is then an easy matter to compute ℓ' in an additional $O(|V| \log |V|)$ steps using binary search. □

THEOREM 3. Algorithm 3 solves the minimum cost network flow problem in $O(|V|U^*)$ pivots, where $U^* = \min(|V|^2 \log |V|, |V| \text{ BIT}(b))$.

PROOF. Let $T^1,\ldots,T^t$ be the trees determined by the "outer while loop". Let $\ell^i = \text{SUPSTRONG}(T^i)$ for $i \in [1..t]$. Because of the condition of the inner while loop, we have $\ell^k < \ell^{k+1}$ for $k \in [1..t-1]$. It is clear that $t \leq |V| \text{ BIT}(b)$. We will next prove that $t \leq |V|^2 + 2|V|^2 \log |V|$.

Let I^k be defined as follows.

$$I^k = \{i \in [i..t] : \text{PERM}(\ell^i) = k\}.$$

Let us partition I^k into I_1^k and I_2^k, where I_1^k consists of the first $|V| = m+1$ elements of I^k. By the pigeon hole principle, there are integers $i,j \in I_1^k$ such that $\ell^i \equiv \ell^j (\text{mod } m)$. It follows that T^i is not feasible for $\text{PROB}(\ell^j+1)$; hence by Lemma 6, $\ell^p \leq \ell^j + 2|V| \log |V|$ for all $p \in I_2^k$. Therefore, $|I_2^k| \leq 2|V| \log |V|$, and $|I^k| \leq |V| + 2|V| \log |V|$. Since $[1..t]$ partitions into $I^1,\ldots,I^m$, it follows that $t \leq |V|^2 + 2|V|^2 \log |V|$.

Now let $T^1,\ldots,T^p$ be a sequence of trees determined by the inner while loop. To complete the proof of Theorem 3, it suffices to show that $p \leq |V|$.

We first note that each of the pivots is degenerate with respect to $\text{PROB}(\ell+1)$, where $\ell = \text{SUPSTRONG}(T^1)$. Moreover, if $T = T^k$, then $x^T(\ell+1) = x(\ell+1)$, i.e., the tree T is optimal for $\text{PROB}(\ell+1)$.

We will soon apply the results of Lemma 7. However, first we will prove that the number of non-root "source vertices" in T^1 is exactly one.

Let $T = T^1$. Since T is strongly feasible for PROB(ℓ) it follows that $x_e^T(\ell) > 0$ for each upward edge e of T.
Let $r \in [1..m]$ with $\ell+1 = sm+r$. Then $x_e^T(\ell+1)$ is obtained from $x_e^T(\ell)$ by sending 2^{-s} units of flow along the path $P^T(0,r)$. Since T is strongly feasible for PROB(ℓ) but not for PROB($\ell+1$) we may conclude that

(i) $x_e^T(\ell+1) = 0$ for some upward edge $e \in P^T(0,r)$

(ii) $x_e^T(\ell+1) \geq 2^{-s}$ for each downward edge $e \in P^T(0,r)$

(iii) $x_e^T(\ell+1) \geq 2^{-s}$ for each upward edge $e \notin P^T(0,r)$.

If we now contract the edges of T for which $x_e^T(\ell+1) > 0$ (as in the proof of Lemma 7), the only upward edges would be on the path $P^T(0,r)$. It follows that vertex r is the unique source vertex of the contracted tree.

Let us now apply the results of Lemma 7. Without loss of generality we assume that $x_e(\ell+1) = 0$ for all $e \in E$. Otherwise we would contract those edges e for which $x_e(\ell+1) > 0$, without effecting the pivoting in the inner loop.

Let G^i, S^i, and d^i be defined as in Lemma 7. We have just shown that $|S^1| = 1$. By (iii) and (iv) of Lemma 7, it follows that

$$\sum_{v \in G^i} d^i(v) = |G^i|.$$

By (i) and (ii) of Lemma 7, it follows that

$$G^i \underset{\neq}{\subset} G^{i+1}$$

and thus the number of pivots is at most $|V|$. This completes the proof of Theorem 3. □

5. A Computational Analysis of Algorithm 3.

In this section we will outline why the number of arithmetic operations for our second dual simplex algorithm is $O\big(U^*(|E| + |V| \log |V|)\big)$. In fact, it suffices to show that the "inner while loop" is really a minor speed-up of the usual Dijkstra algorithm.

Rather than give a detailed formal proof of the equivalence of the dual pivots and Dijkstra's steps, we will illustrate a pivot in Figures 1a and 1b and outline the proof of the equivalence.

The Portrayal of a Dual Simplex Pivot

Let T be a spanning tree obtained by the algorithm and let S be the next spanning tree obtained by pivoting out edge a from T and pivoting in edge e. In order to see why the pivot in equivalent to permanently labeling a vertex (or more) in Dijkstra's algorithm, let us review some properties of the dual simplex algorithm.

First let us suppose that T is optimal for PROB(ℓ) but not strongly feasible, and suppose that the inner while loop will terminate with a tree that is strongly optimal for PROB(ℓ). Suppose also that $r \equiv \ell(\text{mod } m)$ for vertex $r \in [i..m]$.

Next, let us contract all the edges of T and S for which the flow is positive. As mentioned in the previous section, these edges will not be pivoted out until after obtaining a tree that is strongly feasible. Recall that each of these edges $(i,j) \in T$ with $x_{ij}(\ell) > 0$ induces two edges (i,j) and (j,i) in the auxiliary graph for PROB(ℓ) each with a reduced cost of 0. Thus contracting such an edge in G correspond to contracting a strongly connected subgraph of 0-length edges in the auxiliary graph. This "preprocessing" is in the shortest path problem for PROB(ℓ) may be implemented in $O(|E|)$ steps.

We now illustrate what such a contracted tree T may look like.

We have portrayed the edges of T-a in Figure 1a as two subtrees T^0 and T^r rooted at vertex 0 and vertex r respectively, and these subtrees are connected by the edge a. Also each edge in T-a is a downward edge of one of the subtrees. To see why each edge of T^0 and T^r is downward, recall from the proof of Thereom 3 that the contracted graph T must have at most two source vertices: vertex 0 and vertex r. Moreover, if we delete edge a then vertex 0 and vertex r are both sources. It follows that each edge of T^0 and T^r is downward.

Each path $P^T(0,v)$ for $v \in T^0$ is the shortest path in the auxiliary graph for PROB(ℓ) since each edge of the path has a reduced cost of zero and all other costs are non-negative. Thus T^0 corresponds to a set of "permanently labeled" vertices of Dijkstra's algorithm.

The edge e is the minimum cost edge directed from a vertex in T^0 (i.e., a "permanently labeled vertex") to a vertex in T^r (i.e., an "unlabeled vertex"). Indeed, $F(T,a) = \delta(T^0,T^r)$.

To pivot from T to S, as in Figure 1b, we pivot out edge a and pivot in edge e. With respect to the data structures representing the tree, we need only to change the predecessor of the head of e, say vertex v. If u was the predecessor of v in T^r, then (u,v) is the edge to be pivoted out of S. If the head of e were r, then S would be a strongly optimal basis. We observe that we are "permanently labeling" the subtree of T^r rooted at vertex v. In this sense, the dual simplex algorithm offers a speed-up of the usual Dijkstra steps.

In conclusion, we can find a strongly optimal basis for PROB(ℓ) in $O(|E| + |V| \log |V|)$ steps using a minor modification of Fredman and Tarjan's implementation of Dijkstra's algorithm to solve the shortest path problem. The major difference here is that we always maintain a dual feasible basis, and we may "permanently label" a number of vertices in a single step.

Incidentally, there would be no need to calculate the reduced costs and the dual prices at each step. We can defer these calculations until after finding the strongly optimal basis for PROB(ℓ).

We are currently investigating how fast this dual simplex algorithm is in practice. So far, the results are too preliminary to report. However, we can add that other researchers have reported favorable computation times using Edmonds-Karp scaling technique. For example, Ikura and Nemhauser (1984) have successfully applied the scaling technique to the transportation problem. Gabow (1984) has also applied scaling to the shortest path problem and the optimal assignment problem with very good computational results.

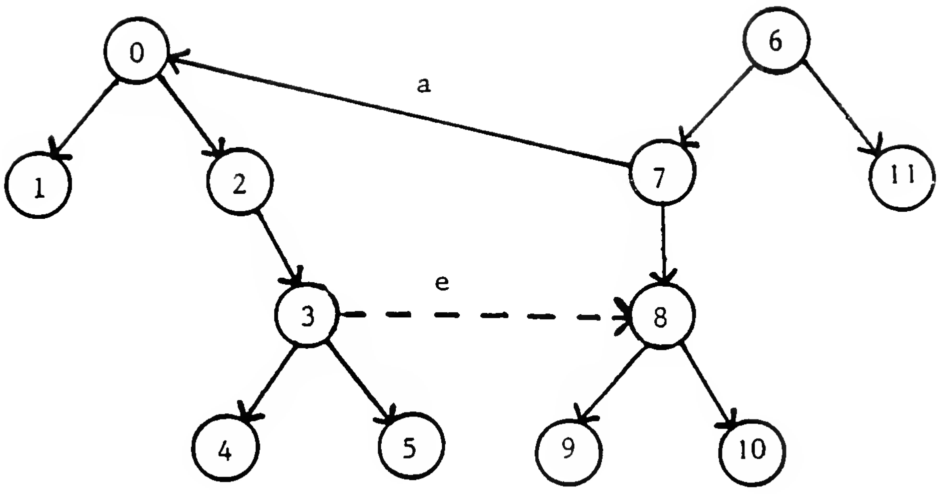

Figure 1a. A tree T. T-a forms two directed trees, T^0 and T^6 rooted at vertex 0 and vertex 6 respectively. (r=6).

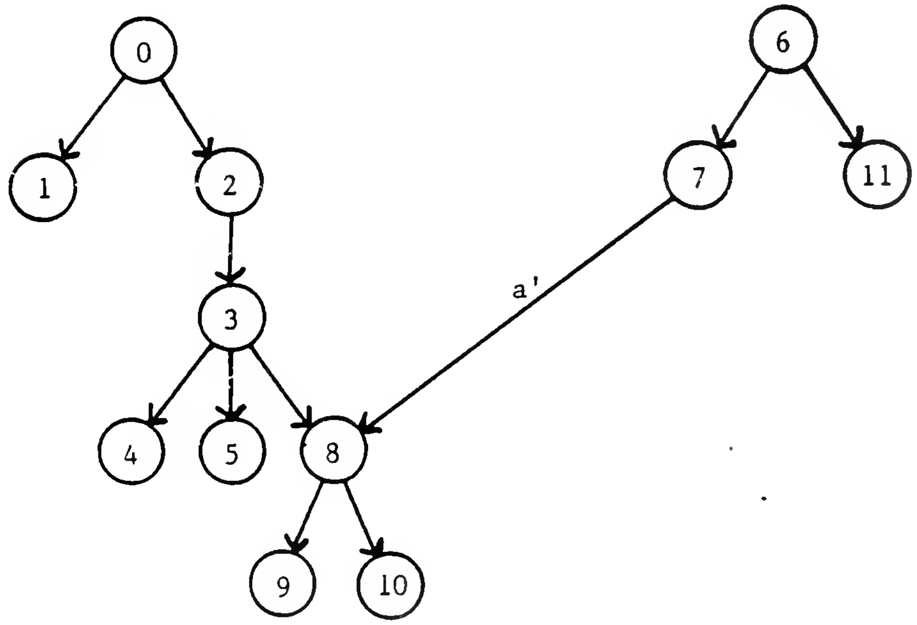

Figure 1b. The tree S obtained from the tree in Figure 1a after pivoting out edge a and pivoting in edge e. Edge a' is the next edge to be pivoted out.

Acknowledgements

I wish to thank Jack Edmonds, the associate editor, and referees for their helpful comments on an earlier draft of this paper.

I also wish to acknowledge partial support provided through ONR grant N00014-83-K-0220.

References

Ali, A., R. Helgason, J. Kennington and H. Lall (1978). Primal simplex network codes: state of the art implementation technology. Networks 8, 315-339.

Balinski, M. (1982). Signature methods for the assignment problem. To appear in Operations Research.

Balinski, M. (1983). Signatures des points extrêmes du polyhedre dual du problème de transport. C.R. Acad. Sc. Paris, Vol. 296, Series I, 457-459.

Balinski, M. (1984). The Hersh conjecture for dual transportation polyhedra. To appear in Mathematics of Operations Research.

Barr, R., F. Glover and D. Klingman (1977). The alternating basis algorithm for assignment problems. Mathematical Programming 13, 1-13.

Cunningham, W. (1976). A network simplex method. Mathematical Programming 11, 105-116.

Cunningham, W. (1979). Theoretical properties of the network simplex method. Mathematics of Operations Research 4, 196-208.

Dantzig, G.B. (1963). Linear Programming and Extensions. Princeton University Press. Princeton, N.J.

Edmonds, J. (1970), unpublished manuscript.

Edmonds, J. and R. Karp (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19, 248-264.

Fredman, M. and R. Tarjan (1984). Fibonacci heaps and their uses in improved network optimization algorithms. Unpublished manuscript.

Gabow, H.N. (1984). On the Theoretic and Practical Efficiency of Scaling Algorithms for Network Problems. Presented at the 1984 ORSA/TIMS conference in Dallas.

Glover, F. and D. Klingman (1976). A practitioner's guide to the state of large scale and network-related problems. AFIPS proceedings, Volume 45, 945-950.

Hung, M. (1983). A polynomial simplex method for the assignment problem. _Operations Research_ 31, 595-600.

Ikura, Y. and G. Nemhauser (1983). A polynomial-time dual simplex algorithm for the transportation problem. Technical report No. 602, SOR & IE, Cornell University, September 1983.

Karp, R. and J. Orlin (1981). Parametric shortest path algorithms with an application to cyclic staffing. _Discrete Applied Mathematics_ 3, 37-45.

Karp, R., J.K. Lenstra, C. McDiarmid, and A. Rinnooy Kan (1984). Probabilistic analysis of combinatorial algorithms: an annotated bibliography. Report OS-128411. Centrum voor Wiskunde en Informatica. Amsterdam, The Netherlands.

Megiddo, N. (1981). Towards a genuinely polynomial algorithm for linear programming. _SIAM J. Comput._ 12, 347-353.

Orlin, J. (1984). On the simplex algorithm for networks and generalized networks. To appear in _Mathematical Programming_.

Roohy-Laleh, E. (1980). Improvements to the theoretical efficiency of the network simplex method. PhD Dissertation, Carleton University.

Tardos, E. (1984a). A strongly polynomial minimum cost circulation algorithm. To appear in _Combinatorica_.

Tardos, E. (1984b). Work in progress.

Zadeh, N. (1973). A bad network flow problem for the simplex method and other minimum cost flow algorithms. _Mathematical Programming_ 5, 255-266.

3790 U13 [illegible]

For [illegible]
[illegible]
[illegible]